

ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH

TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

KHOA CÔNG NGHỆ THÔNG TIN



# ĐỒ ÁN 01 MÔN KIẾN TRÚC MÁY TÍNH VÀ HỢP NGỮ

Đề tài:

## BIỂU DIỄN VÀ TÍNH TOÁN SỐ HỌC TRONG MÁY TÍNH

Người thực hiện:

Nguyễn Hoàng Thái Dương (18120336)

Trần Thanh Quang (18120230)

Trương Trọng Lộc (18120197)

Giảng viên hướng dẫn:

Thầy ThS. Lê Viết Long

Thành phố Hồ Chí Minh – Tháng 05, 2020

## MỤC LỤC

MỤC LỤC.....	2
1. THÀNH VIÊN VÀ PHÂN CHIA CÔNG VIỆC: .....	5
2. BIỂU DIỄN SỐ NGUYÊN LỚN 128 BIT .....	5
2.1. Thiết kế cấu trúc dữ liệu QInt:.....	5
2.2. Chuyển đổi giữa các hệ cơ số: .....	5
2.2.1. Ý tưởng chuyển đổi:.....	5
2.2.2. Nhóm hàm SetBit, GetBit: .....	5
2.2.3. Chuyển đổi từ chuỗi nhị phân (string) sang QInt:.....	6
2.2.4. Chuyển đổi từ chuỗi thập phân (string) sang QInt:.....	7
2.2.5. Chuyển đổi từ chuỗi thập lục phân (string) sang QInt:.....	7
2.2.6. Chuyển đổi từ chuỗi ở hệ cơ số 2, 10, 16 (string) sang QInt: .....	7
2.2.7. Chuyển đổi từ QInt sang chuỗi nhị phân (string):.....	7
2.2.8. Chuyển đổi từ QInt sang chuỗi thập phân (string):.....	7
2.2.9. Chuyển đổi từ QInt sang chuỗi thập lục phân (string):.....	8
2.2.10. Chuyển đổi từ QInt sang chuỗi ở hệ cơ số 2, 10, 16 (string): .....	8
2.2.11. Chuyển đổi từ chuỗi ở hệ cơ số a sang chuỗi ở hệ cơ số b (a, b = 2, 10, 16): .....	8
2.3. Các nhóm toán tử:.....	8
2.3.1. Toán tử cộng: .....	8
2.3.2. Toán tử trừ:.....	9
2.3.3. Toán tử nhân: .....	9
2.3.4. Toán tử chia và lấy dư:.....	9
2.3.5. Toán tử so sánh và gán:.....	11

2.3.6. Toán tử AND, OR, XOR, NOT: .....	11
2.3.7. Toán tử dịch trái, dịch phải: .....	11
2.3.8. Toán tử xoay trái, xoay phải: .....	12
3. BIỂU DIỄN SỐ CHẤM ĐỘNG CHÍNH XÁC CAO 128 BIT .....	12
3.1. Thiết kế kiểu dữ liệu QFloat: .....	12
3.2. Chuyển đổi giữa các hệ cơ số: .....	12
3.2.1. Ý tưởng chuyển đổi: .....	12
3.2.2. Nhóm hàm SetBit, GetBit: .....	12
3.2.3. Nhóm hàm kiểm tra các trường hợp đặc biệt: .....	12
3.2.4. Chuyển đổi từ chuỗi nhị phân (string) sang QFloat: .....	13
3.2.5. Chuyển đổi từ chuỗi thập phân (string) sang QFloat: .....	13
3.2.6. Chuyển đổi từ chuỗi ở hệ cơ số 2, 10 (string) sang QFloat: .....	14
3.2.7. Chuyển đổi từ QFloat sang chuỗi nhị phân (string): .....	14
3.2.8. Chuyển đổi từ QFloat sang chuỗi thập phân (string): .....	15
3.2.9. Chuyển đổi từ QFloat sang chuỗi ở hệ cơ số 2, 10 (string): .....	15
3.3. Các nhóm toán tử: .....	16
3.3.1. Toán tử cộng: .....	16
3.3.2. Toán tử trừ: .....	16
3.3.3. Toán tử nhân: .....	17
3.3.4. Toán tử chia: .....	17
4. PHẠM VI BIỂU DIỄN CỦA CÁC KIỂU DỮ LIỆU ĐÃ THIẾT KẾ .....	18
4.1. Kiểu số nguyên lớn QInt 128 bit: .....	18
4.2. Kiểu số chấm động chính xác cao: .....	18
5. GIAO DIỆN NGƯỜI DÙNG .....	19

5.1. Chế độ QInt: .....	19
5.2. Chế độ QFloat:.....	20
6. GIAO DIỆN CHƯƠNG TRÌNH ỨNG VỚI TEST CASE.....	20
6.1. Tham số dòng lệnh: .....	20
6.1.1. Kiểu dữ liệu QInt: .....	20
6.1.2. Kiểu dữ liệu QFloat:.....	25
6.2. Giao diện người dùng: .....	31
7. ĐÁNH GIÁ MỨC ĐỘ HOÀN THÀNH .....	32
8. TÀI LIỆU THAM KHẢO.....	33

## 1. THÀNH VIÊN VÀ PHÂN CHIA CÔNG VIỆC:

STT	MSSV	Họ và tên	Công việc
1	18120336	Nguyễn Hoàng Thái Dương	Giao diện Calculator, phép nhân, chia QFloat.
2	18120230	Trần Thanh Quang	Chuẩn bị 296 bộ testcase, nhập xuất QInt, QFloat, các toán tử so sánh, AND, OR, XOR, NOT, phép dịch trái, dịch phải, xoay trái, xoay phải QInt.
3	18120197	Trương Trọng Lộc	Chuyển đổi các hệ cơ số QInt và QFloat, toán tử cộng, trừ, nhân, chia, chia lấy dư QInt và cộng, trừ QFloat.

## 2. BIỂU DIỄN SỐ NGUYÊN LỚN 128 BIT

### 2.1. Thiết kế cấu trúc dữ liệu QInt:

- Số nguyên lớn QInt bao gồm 128 bit được lưu trữ bởi mảng *data* gồm 4 phần tử kiểu số nguyên. Ở phương thức khởi tạo mặc định, tất cả 128 bit này đều là bit 0.
- Vì 1 phần tử kiểu số nguyên có kích thước 4 bytes nên ta sử dụng 4 phần tử để biểu diễn (thỏa yêu cầu 16 bytes như đề bài).

### 2.2. Chuyển đổi giữa các hệ cơ số:

#### 2.2.1. Ý tưởng chuyển đổi:

Đầu vào sẽ là một chuỗi String ở dạng nhị phân / thập lục phân / thập phân. Sau đó ta lưu chuỗi String này vào QInt bằng các hàm chuyển đổi. Từ QInt ta chuyển sang lại dạng chuỗi ở các dạng nhị phân / thập lục phân / thập phân. Như vậy, ta có thể chuyển từ chuỗi String từ hệ cơ số *a* sang chuỗi String ở hệ cơ số *b* dựa vào trung gian là QInt.

#### 2.2.2. Nhóm hàm SetBit, GetBit:

- Lưu ý về việc lưu các dãy bit. Ta xét ví dụ sau: Giả sử ta có chuỗi 8 bit. Khi đó, đối với dạng chuỗi (string), vị trí từng bit, lúc này mỗi bit là phần tử của string nên sẽ được

đánh số thứ tự từ trái sang phải. Tuy nhiên, khi lưu trữ dãy bit, ta đánh số từ phải sang trái. Giả sử ta lưu dãy 8 bit này vào 1 mảng gồm 2 phần tử có kích thước 4 bit, ta lưu:

Vị trí bit (dạng string)	0	1	2	3	4	5	6	7
Vị trí bit (khi lưu trữ)	7	6	5	4	3	2	1	0
Mảng 2 phần tử (mỗi phần tử 4 bit)	A[0]				A[1]			
Vị trí bit ở mỗi phần tử trong mảng	3	2	1	0	3	2	1	0

- Để dễ hiểu, ta xét ví dụ, dãy 8 bit 1100 1011 ở dạng chuỗi (string), khi đó phần tử `str[0] = '1'` sẽ tương ứng với vị trí bit thứ 7 khi lưu trữ. Khi đó bit này sẽ được lưu trữ vào `A[1]` ở vị trí thứ 3. Sở dĩ là `A[1]` vì bit lưu trữ ở vị trí thứ 7 chia cho kích thước một phần tử là 4, lấy phần nguyên ta được 1; ở vị trí thứ 3 trong `A[1]` thì 7 chia 4 lấy phần dư là 3. Từ đó, ta thấy với dãy 1100 1011 thì `A[0]` gồm dãy 4 bit 1011 và `A[1]` gồm dãy 4 bit 1100.
- Tổng quát để xác định vị trí bit lưu ở đâu trong `QInt` dựa vào vị trí bit khi lưu trữ, ta làm như sau:
  - o Xác định *block* dựa vào vị trí bit khi lưu trữ bằng cách lấy vị trí đó chia cho kích thước của một *block*. Với `QInt`, mỗi *block* là kiểu số nguyên nên kích thước là 32 bit.
  - o Xác định vị trí bit trong *block* bằng cách lấy vị trí bit khi lưu trữ chia lấy dư cho kích thước của một *block*.
- Sau khi xác định được vị trí bit trong `QInt`, ta tiến hành `GetBit` hoặc `SetBit` như sau:
  - o Với `GetBit`, ta thực hiện phép AND với 1 tại bit đó.
  - o Với `SetBit`, nếu là bit 1, ta thực hiện phép OR với 1 tại bit đó (vì OR với 1 sẽ ra 1); nếu là bit 0, ta thực hiện phép AND với 0 tại bit đó (vì AND với 0 sẽ ra 0)

### 2.2.3. Chuyển đổi từ chuỗi nhị phân (string) sang `QInt`:

Ta lần lượt duyệt qua từng phần tử của chuỗi string. Cứ mỗi lần duyệt ta gọi hàm `SetBit` để lưu vào `QInt`. Ở đây, ta duyệt từ đầu đến cuối của chuỗi string, nên khi `SetBit` vị trí sẽ là `str.length() - 1 - i`, với `i` là biến đếm duyệt chuỗi string, `str.length()` là độ dài chuỗi string (vì vị trí set bit là vị trí lưu trữ - sẽ được đánh số thứ tự ngược với chuỗi string như đã trình bày ở trên).

#### 2.2.4. Chuyển đổi từ chuỗi thập phân (string) sang QInt:

- Đầu tiên ta xét chuỗi thập phân biểu thị số âm hay dương, nếu có dấu '-' ta thực hiện xóa dấu này ra khỏi chuỗi và có lưu lại sự thay đổi này bởi biến *sign* kiểu bool để làm cơ sở xét dấu lúc sau.
- Ta thực hiện chuyển chuỗi thập phân sang QInt bằng cách chia chuỗi này cho 2, phần dư sẽ được SetBit vào QInt, còn phần nguyên ta tiếp tục chia cho 2. Lặp lại như vậy cho đến khi chuỗi bằng "0" hoặc chuỗi rỗng.
- Ta dựa vào tình trạng biến *sign*, nếu biến *sign* biểu thị số âm ta thực hiện chuyển sang dạng bù 2 (ta thực hiện đổi bit kết quả tìm được ở trên, sau đó cộng cho 1).

#### 2.2.5. Chuyển đổi từ chuỗi thập lục phân (string) sang QInt:

- Do ở hệ thập lục phân, biểu diễn bởi 16 ký tự là 0 đến 9 và A đến F. Nên trước tiên ta thực hiện chuẩn hóa các ký tự đầu vào thành kí tự in hoa (nếu đầu vào là chữ thường).
- Điểm đặc biệt ở hệ thập lục phân là mỗi ký tự biểu diễn của nó sẽ được biểu diễn bởi 4 bit nhị phân (Vì mỗi ký tự của nó chạy từ 0 đến 15). Vì vậy ta sẽ duyệt lần lượt từng ký tự ở hệ thập lục phân và chuyển nó về dạng 4 bit nhị phân.
- Lưu ý khi chuyển, nếu gặp ký tự chữ ta phải chuyển đó về dạng số tương ứng rồi sau đó mới chuyển về dạng 4 bit. Cụ thể A là 10, B là 11, C là 12, D là 13, E là 14 và F là 15.

#### 2.2.6. Chuyển đổi từ chuỗi ở hệ cơ số 2, 10, 16 (string) sang QInt:

Ta sử dụng câu lệnh *if* trên tham số đầu vào (đầu vào sẽ được truyền vào biểu thị hệ cơ số cần được chuyển đổi vào QInt). Sau đó dựa vào đầu vào ta lần lượt gọi các hàm đã trình bày ở [2.2.3](#), [2.2.4](#) và [2.2.5](#) để lưu trữ vào QInt.

#### 2.2.7. Chuyển đổi từ QInt sang chuỗi nhị phân (string):

Ta thực hiện lấy từng bit trong QInt và cộng vào chuỗi kết quả. Sau đó ta loại bỏ các số 0 thừa ở đầu. Lưu ý giữ lại 1 bit 0 nếu 128 bit đều là 0.

#### 2.2.8. Chuyển đổi từ QInt sang chuỗi thập phân (string):

- Ta thực hiện lấy từng bit trong QInt theo đúng thứ tự lưu trữ (từ 0 đến 126, trừ bit dấu).  
Tại mỗi bit nếu giá trị bit là 1 ta thực hiện cộng với  $2^i$  với  $i$  là số thứ tự của bit.

- Ta xét bit dấu, nếu bit dấu là 1, ta thực hiện phép trừ giữa  $2^{127}$  và kết quả tìm được ở trên, sau đó thêm '-' vào trước chuỗi kết quả.

### 2.2.9. Chuyển đổi từ QInt sang chuỗi thập lục phân (string):

- Đầu tiên, ta tạo kiểu dữ liệu *map* để lưu trữ các ký tự biểu diễn hệ thập lục phân ở dạng 4 bit. Sở dĩ chọn kiểu dữ liệu *map* vì nó dễ dàng truy xuất phần tử với index dạng chuỗi.
- Sau đó ta duyệt lần lượt bộ 4 bit trong QInt từ vị trí  $i = 0$  và dựa vào chuỗi 4 bit đó ta cộng vào chuỗi kết quả ký tự ở hệ thập lục phân tương ứng thông qua kiểu dữ liệu *map*.
- Sau đó ta thực hiện xóa các số 0 thừa (nếu có) và thực hiện đảo ngược lại kết quả (vì  $i = 0$  nên ta phải đảo ngược kết quả).

### 2.2.10. Chuyển đổi từ QInt sang chuỗi ở hệ cơ số 2, 10, 16 (string):

Ta sử dụng câu lệnh *if* trên tham số đầu vào (đầu vào sẽ được truyền vào biểu thị hệ cơ số cần được chuyển đổi ra string từ QInt). Sau đó dựa vào đầu vào ta lần lượt gọi các hàm đã trình bày ở [2.2.7](#), [2.2.8](#) và [2.2.9](#) để chuyển từ QInt sang string.

### 2.2.11. Chuyển đổi từ chuỗi ở hệ cơ số a sang chuỗi ở hệ cơ số b ( $a, b = 2, 10, 16$ ):

Ta sử dụng câu lệnh *if* trên tham số đầu vào (đầu vào sẽ được truyền vào biểu thị hệ cơ số a – hệ cơ số cần phải chuyển đổi và hệ cơ số b – sau khi chuyển đổi). Sau đó dựa vào đầu vào ta lần lượt gọi các hàm đã trình bày ở [2.2.6](#) và [2.2.10](#) để thực hiện việc chuyển đổi.

## 2.3. Các nhóm toán tử:

### 2.3.1. Toán tử cộng:

Phép cộng 2 số QInt được thực hiện dựa trên nguyên tắc cộng bit. Ta cộng lần lượt các bit theo thứ tự từ phải sang trái. Trong quá trình cộng có biến *carry* để lưu lại bit nhớ. Ta cộng bit ở số QInt trên với bit tương ứng ở số QInt dưới với bit nhớ ta được kết quả lưu ở biến *tmp*, sau đó ta thực hiện SetBit bằng cách lấy kết quả vừa tìm lấy dư cho 2, đồng thời cập nhật biến nhớ bằng cách lấy *tmp* chia cho 2.

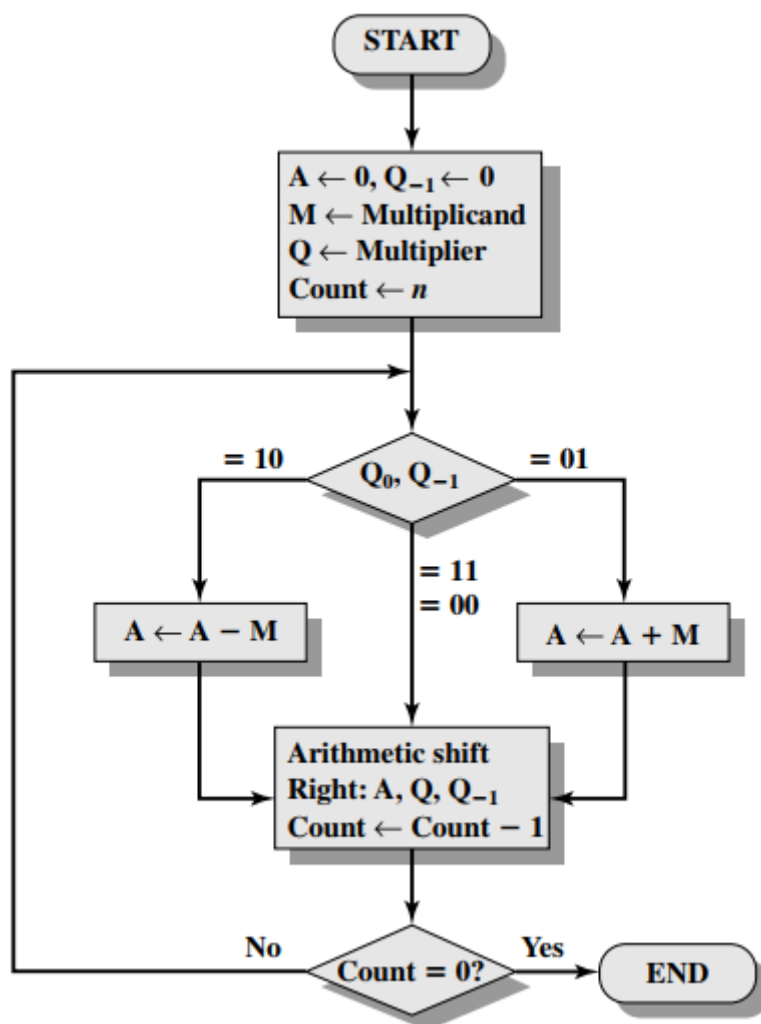


### 2.3.2. Toán tử trừ:

Ta thực hiện chuyển số trừ sang dạng bù 2 (đổi bit và cộng 1), sau đó thực hiện phép cộng giữa số bị trừ và số trừ ở dạng bù 2.

### 2.3.3. Toán tử nhân:

Ta thực hiện theo thuật toán Booth. Lưu đồ thuật toán Booth sau đây được trích từ Chương 9, sách Computer Organization and Architecture Designing for performance (8<sup>th</sup> edition) của William Stallings.

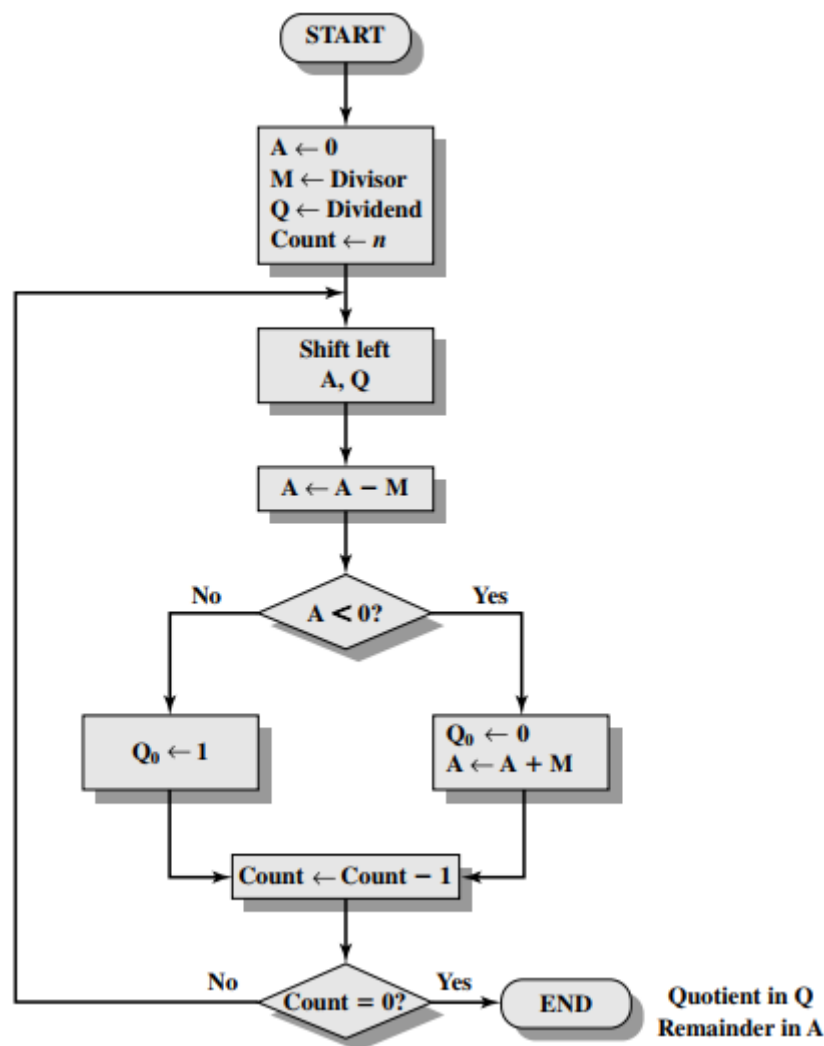


### 2.3.4. Toán tử chia và lấy dư:

- Phép chia và lấy dư được thực hiện theo thuật toán chia không dấu. Nếu số bị chia hay số chia là số âm, ta thực hiện chuyển các số âm đó sang dạng bù 2 để thực hiện chia

không dấu. Sau khi chia không dấu ta dựa vào dấu của số bị chia và số chia để xác định dấu của thương và số dư cho phù hợp.

- Ta lưu ý về thương và số dư. Gọi D là số bị chia, V là số chia, Q là thương, R là số dư. Ta có biểu thức sau:  $D = Q \times V + R$ . Dấu của R chính là dấu của D. Dấu của Q chính là dấu của D nhân với dấu của V. Xét ví dụ sau để hiểu rõ:
  - $D = 7, V = 3 \Rightarrow Q = 2, R = 1$
  - $D = -7, V = 3 \Rightarrow Q = -2, R = -1$
  - $D = 7, V = -3 \Rightarrow Q = -2, R = 1$
  - $D = -7, V = -3 \Rightarrow Q = 2, R = -1$
- Lưu đồ thuật toán chia không dấu sau đây được trích từ Chương 9, sách Computer Organization and Architecture Designing for performance (8<sup>th</sup> edition) của William Stallings.



**2.3.5. Toán tử so sánh và gán:**

- Phép  $>$ : Kiểm tra xem 2 số có cùng dấu hay không, nếu khác dấu, ta kiểm tra số đầu tiên nếu là âm sẽ return false, ngược lại return true. Nếu cùng dấu, ta lần lượt so sánh các bit theo thứ tự từ bit kế bit dấu đến bit thứ 0 trong vị trí lưu trữ. Nếu bit tại vị trí  $i$  ở số thứ nhất nhỏ hơn bit ở số thứ 2, ta trả về bit dấu của số thứ nhất. Nếu bit tại vị trí  $i$  ở số thứ 1 lớn hơn bit ở số thứ 2, ta đổi bit dấu của số thứ nhất (lấy NOT) và trả về.
- Phép  $==$ : Nếu hai số có bit nào khác nhau thì trả về false, ngược lại trả về true.
- Phép  $!=$ : Nếu hai số không bằng nhau thì return true, ngược lại return false.
- Phép  $<$ : Nếu số thứ nhất không lớn hơn và không bằng số thứ 2 return true; ngược lại return false
- Phép  $>=$ : Nếu số thứ nhất không nhỏ hơn số thứ 2 thì return true, ngược lại return false.
- Phép  $<=$ : Nếu số thứ nhất không lớn hơn số thứ 2 thì return true, ngược lại return false.
- Toán tử gán  $=$ : Nếu 2 địa chỉ của 2 biến khác nhau ta thực hiện gán thuộc tính của biến này sang biến khác.

**2.3.6. Toán tử AND, OR, XOR, NOT:**

- Ở các toán tử AND, OR, XOR, ta thực hiện trên từng cặp bit của 2 số QInt.
- Ở toán tử NOT, ta đổi từng bit của số đó.
- Cụ thể:
  - Ở phép AND, nếu cả 2 bit của 2 số QInt là 1 ta SetBit 1 vào số QInt kết quả, ngược lại SetBit 0.
  - Ở phép OR, nếu cả 2 bit của 2 số QInt là 0 ta SetBit 0 vào số QInt kết quả, ngược lại SetBit 1.
  - Ở phép XOR, nếu 2 bit của 2 số QInt bằng nhau, ta SetBit 0 vào số QInt kết quả, ngược lại SetBit 1.
  - Ở phép NOT, ta thực hiện đổi từng bit (từ 0 sang 1 và từ 1 sang 0).

**2.3.7. Toán tử dịch trái, dịch phải:**

- Ta truyền vào tham số là số lượng bit cần dịch.
- Ở đây ta thực hiện phép dịch số học. Khi dịch trái ta thêm các bit 0 vào cuối dãy bit sau khi dịch. Khi dịch phải ta thêm các bit dấu vào đầu dãy bit sau khi dịch.

### 2.3.8. Toán tử xoay trái, xoay phải:

- Ta truyền vào tham số là số lượng bit cần xoay.
- Khi xoay trái / xoay phải  $x$  bit, ta lấy số lượng bit cần xoay bằng cách chia lấy dư với 128 (vì phép xoay sẽ thực hiện tuần hoàn với chu kỳ 128 bit).
- Mỗi lần xoay ta dịch trái / dịch phải 1 bit, lặp lại cho đến khi đủ  $x$  lần.

## 3. BIỂU DIỄN SỐ CHẤM ĐỘNG CHÍNH XÁC CAO 128 BIT

### 3.1. Thiết kế kiểu dữ liệu QFloat:

- Số chấm động chính xác cao QFloat bao gồm 128 bit được lưu trữ bởi mảng data gồm 4 phần tử kiểu số nguyên. Ở phương thức khởi tạo mặc định, tất cả 128 bit này đều là bit 0.
- Vì 1 phần tử kiểu số nguyên có kích thước 4 bytes nên ta sử dụng 4 phần tử để biểu diễn (thỏa yêu cầu 16 bytes như đề bài).
- Trong 128 bit, có 1 bit biểu diễn dấu, 15 bit biểu diễn phần mũ và 112 bit biểu diễn phần định trị.

### 3.2. Chuyển đổi giữa các hệ cơ số:

#### 3.2.1. Ý tưởng chuyển đổi:

Đầu vào sẽ là một chuỗi String ở dạng nhị phân / thập phân. Sau đó ta lưu chuỗi String này vào QFloat bằng các hàm chuyển đổi. Từ QFloat ta chuyển sang lại dạng chuỗi ở các dạng nhị phân / thập phân. Như vậy, ta có thể chuyển từ chuỗi String từ hệ cơ số  $a$  sang chuỗi String ở hệ cơ số  $b$  dựa vào trung gian là QFloat.

#### 3.2.2. Nhóm hàm SetBit, GetBit:

Ý tưởng và phương pháp thực hiện giống ở mục [2.2.2](#).

#### 3.2.3. Nhóm hàm kiểm tra các trường hợp đặc biệt:

- bool CheckZero() const:; Hàm kiểm tra có phải QFloat là 0 hay không. Nếu có bất kỳ bit nào trong 127 bit (trừ bit dấu) khác bit 0, return false.

- `bool CheckDenormalized() const`;: Hàm kiểm tra dạng số không chuẩn. Số không chuẩn là số có phần mũ toàn bit 0, và phần trị khác 0.
- `bool CheckNaN() const`;: Hàm kiểm tra có phải là dạng NaN không. Dạng NaN khi phần mũ toàn bit 1, và phần trị khác 0.
- `bool CheckInf() const`;: Hàm kiểm tra có phải là dạng Infinity không. Dạng Infinity khi phần mũ toàn bit 1, và phần trị toàn bit 0.

### 3.2.4. Chuyển đổi từ chuỗi nhị phân (string) sang QFloat:

Ta lần lượt duyệt qua từng phần tử của chuỗi string. Cứ mỗi lần duyệt ta gọi hàm `SetBit` để lưu vào QFloat. Ở đây, ta duyệt từ đầu đến cuối của chuỗi string, nên khi `SetBit` vị trí sẽ là `str.length() - 1 - i`, với `i` là biến đếm duyệt chuỗi string, `str.length()` là độ dài chuỗi string (vì vị trí set bit là vị trí lưu trữ - sẽ được đánh số thứ tự ngược với chuỗi string như đã trình bày ở trên).

### 3.2.5. Chuyển đổi từ chuỗi thập phân (string) sang QFloat:

- Đầu tiên, ta xét dạng chuỗi thập phân nhập vào có phải dạng “0.0” hoặc “0” hay không, nếu có ta trả về phương thức khởi tạo mặc định của QFloat.
- Ta xét chuỗi thập phân biểu thị số âm hay dương, nếu có dấu ‘-’ ta thực hiện xóa dấu này ra khỏi chuỗi và ta thực hiện `SetBit` dấu vào kết quả. Lưu ý bit dấu là vị trí thứ 127 như đã trình bày ở những phần trên.
- Ta thực hiện phân tách phần nguyên và phần thập phân của chuỗi đầu vào nhờ việc tìm kiếm dấu “.” trong chuỗi đầu vào. Tuy nhiên, một số trường hợp chuỗi đầu vào không có dấu “.”, ta thực hiện thêm vào sau chuỗi thập phân đầu vào “0”.
- Ta thực hiện đổi phần nguyên và phần thập phân của chuỗi đầu vào thành dãy bit nhị phân. Lưu ý, khi đổi phần thập phân thành các dãy bit, ta có các trường hợp sau đây:
  - o Nếu phần nguyên khác 0, nghĩa là số này có dạng 1.F, đây là dạng chuẩn, nên ta thực hiện lấy phần thập phân nhân cho 2, và lưu lại phần nguyên. Sau đó lặp lại việc này cho đến khi đủ bit. Trong quá trình lặp lại, khi đạt được giá trị 1.0 ta thực hiện lưu lại tình trạng này để hỗ trợ việc làm tròn sau này. Đồng thời, giá trị phần mũ được tính như sau:  $exp = intPartBit.size() - 1 + BIAS$  trong đó  $exp$  là giá trị phần mũ,  $intPartBit.size()$  là chiều dài dãy bit nhị phân của phần nguyên,  $BIAS$  trong trường hợp QFloat là  $2^{14} - 1 = 16383$ . Sở dĩ có cách tính

này là vì phần nguyên khác 0 nên tối thiểu là  $\text{length intPartBit} = 1$  nên số lần dời dấu chấm là  $\text{intPartBit.size()} - 1$ .

- Nếu phần nguyên bằng 0, ta có 2 trường hợp: số chuẩn hóa được và số không thể chuẩn hóa. Ta cũng thực hiện lấy phần thập phân nhân cho 2, và lưu lại phần nguyên. Sau đó lặp lại việc này cho đến khi đạt được giá trị 1.0.  $\text{exp}$  được tính bằng  $\text{BIAS}$  trừ đi số lần lặp. Nếu số lần lặp để tìm được giá trị 1.0 nhỏ hơn  $\text{BIAS}$  thì đây là trường hợp số có thể chuẩn hóa được, ta tiếp tục thực hiện nhân phần thập phân cho 2 và lưu lại phần nguyên cho đến khi đủ bit. Ngược lại là dạng số không thể chuẩn hóa, ta thực hiện reset lại chuỗi nhị phân của phần thập phân và nhân phần thập phân cho 2 và lưu lại phần nguyên và tăng  $\text{exp}$  lên 1. Lặp lại cho đến khi  $\text{exp} = 0$ . Trong quá trình lặp lại, khi đạt được giá trị 1.0 ta thực hiện lưu lại tình trạng này để hỗ trợ việc làm tròn sau này.
- Nếu ta chưa tìm được giá trị 1.0 khi lặp lại việc nhân phần thập phân cho 2. Ta tiếp tục lặp lại việc này thêm 3 lần, để thực hiện việc làm tròn theo nguyên tắc *round to the nearest; ties to even*.
- Ta làm tròn phần nguyên: khi thập phân toàn bit 1, ta cộng 1 vào  $\text{exp}$  và set lại phần thập phân là 0. Ví dụ: 10.1111  $\rightarrow$  làm tròn: 11.0000.
- Nếu tổng bit nhị phân phần nguyên và phần thập phân chưa đủ 112 bit ta tiến hành thêm các bit 0 cho đủ bit.
- Sau cùng, ta đổi  $\text{exp}$  sang dãy bit nhị phân và tiến hành SetBit cho  $\text{exp}$  và phần trị vào kết quả.

### 3.2.6. Chuyển đổi từ chuỗi ở hệ cơ số 2, 10 (string) sang QFloat:

Ta sử dụng câu lệnh *if* trên tham số đầu vào (đầu vào sẽ được truyền vào biểu thị hệ cơ số cần được chuyển đổi vào QFloat). Sau đó dựa vào đầu vào ta lần lượt gọi các hàm đã trình bày ở [3.2.4](#) và [3.2.5](#) để lưu trữ vào QFloat.

### 3.2.7. Chuyển đổi từ QFloat sang chuỗi nhị phân (string):

Ta thực hiện lấy từng bit trong QFloat và cộng vào chuỗi kết quả. Lưu ý thứ tự khi GetBit và cộng vào chuỗi. Ta thực hiện GetBit từ bit đầu (bit thứ 127) về bit thứ 0.

**3.2.8. Chuyển đổi từ QFloat sang chuỗi thập phân (string):**

- Ta kiểm tra QFloat có phải là 0 hay dạng NaN hay không, nếu phải ta thực hiện return chuỗi báo tương ứng.
- Ta lưu lại bit dấu của QFloat (bit thứ 127).
- Ta kiểm tra QFloat có phải dạng Infinity hay không, nếu phải ta xét bit dấu và return Inf hay -Inf cho phù hợp.
- Nếu QFloat không nằm trong các dạng đặc biệt trên, ta thực hiện:
  - o Tách phần mũ và phần trị từ QFloat (ta chuyển QFloat sang chuỗi nhị phân cho dễ thao tác).
  - o Ta tính giá trị *exp* bằng cách lấy giá trị của dãy bit – *BIAS*. Với số không chuẩn, phần mũ bằng 0 và  $exp = -16382$  và có dạng 0.F. Với số bình thường, *exp* được tính theo công thức trên và có dạng 1.F.
  - o Dời dấu chấm động khi  $exp \neq 0$ . *exp* dương thì ta dời qua phải. Mỗi lần dịch, phần nguyên cộng thêm bit từ phần thập phân chuyển qua và phần thập phân xóa bớt. Nếu đã dịch hết phần thập phân, ta thêm 0 vào phần nguyên. Sau đó, giảm *exp* đi 1 đơn vị. Ngược lại ta dời qua trái. Mỗi lần dịch, phần thập phân cộng thêm bit từ phần nguyên chuyển qua và phần nguyên xóa bớt bit. Nếu đã dịch hết phần nguyên, ta thêm 0 vào phần thập phân. Sau đó, tăng *exp* lên 1 đơn vị.
  - o Chuyển phần nguyên về dạng thập phân (hệ 10)
  - o Chuyển phần thập phân về dạng thập phân (hệ 10). Khi duyệt bit ở phần này, tại mỗi bit, nếu là bit 1 ta thực hiện cộng với  $2^{-n}$ , với *n* là vị trí bit, tính từ 1.

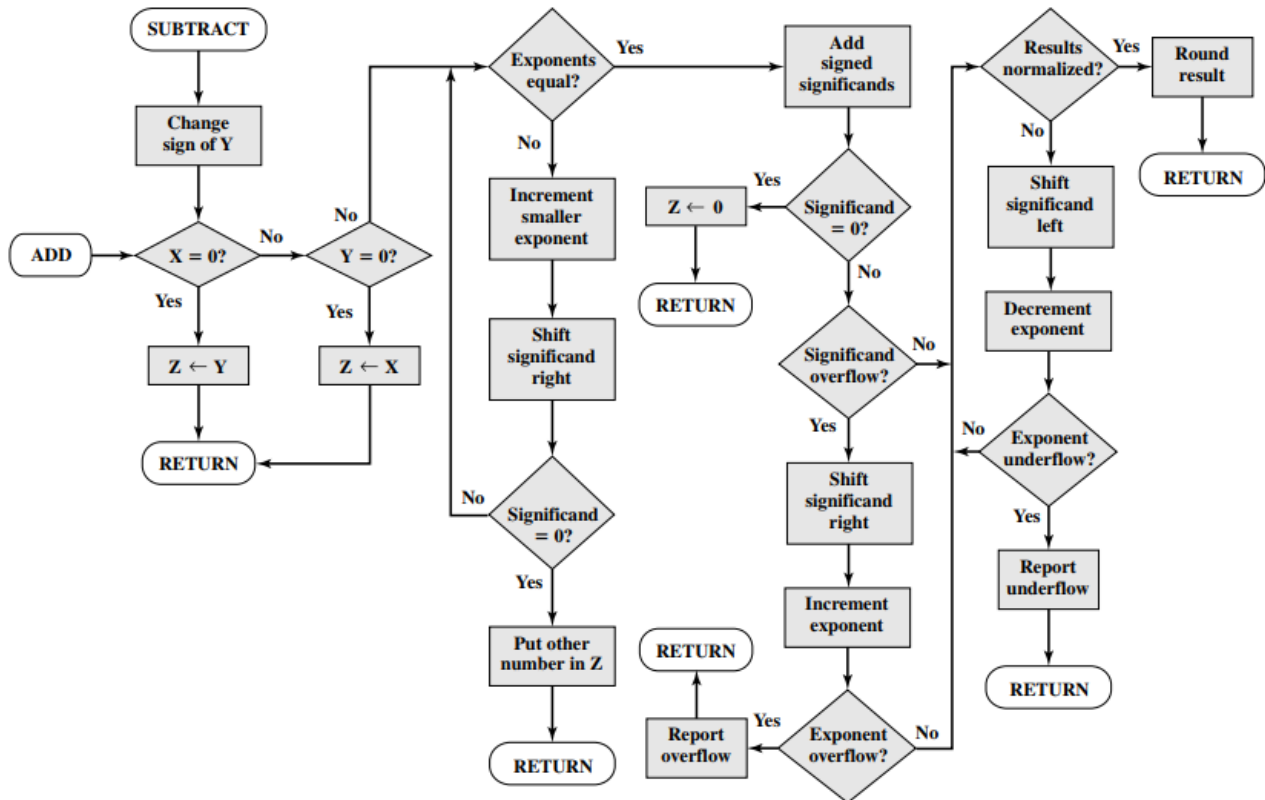
**3.2.9. Chuyển đổi từ QFloat sang chuỗi ở hệ cơ số 2, 10 (string):**

Ta sử dụng câu lệnh *if* trên tham số đầu vào (đầu vào sẽ được truyền vào biểu thị hệ cơ số cần được chuyển đổi ra string từ QFloat). Sau đó dựa vào đầu vào ta lần lượt gọi các hàm đã trình bày ở [3.2.7](#) và [3.2.8](#) để chuyển từ QFloat sang string.

### 3.3. Các nhóm toán tử:

#### 3.3.1. Toán tử cộng:

Ta thực hiện cộng theo lưu đồ thuật toán sau đây được trích từ Chương 9, sách Computer Organization and Architecture Designing for performance (8<sup>th</sup> edition) của William Stallings.



- Ta có thể tóm gọn sơ đồ trong 4 bước chính:
  - Kiểm tra phép cộng với NaN, Infinity và 0.
  - Điều chỉnh phần mũ cho 2 số bằng nhau.
  - Thực hiện cộng có dấu phần định trị của 2 số.
  - Chuẩn hóa kết quả (normalize result).

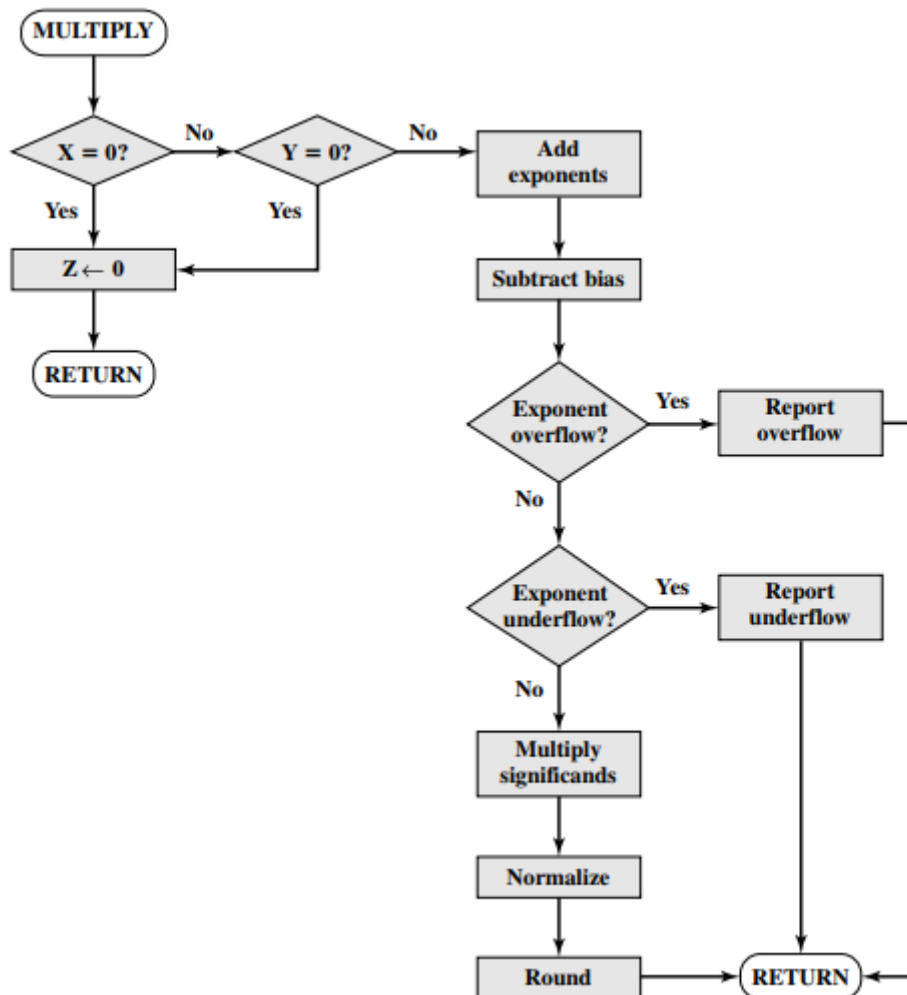
#### 3.3.2. Toán tử trừ:

Ta thực hiện đổi số trừ sang số đối của nó (thực hiện đổi dấu) và thực hiện phép cộng (vì  $a - b = a + (-b)$ ).



### 3.3.3. Toán tử nhân:

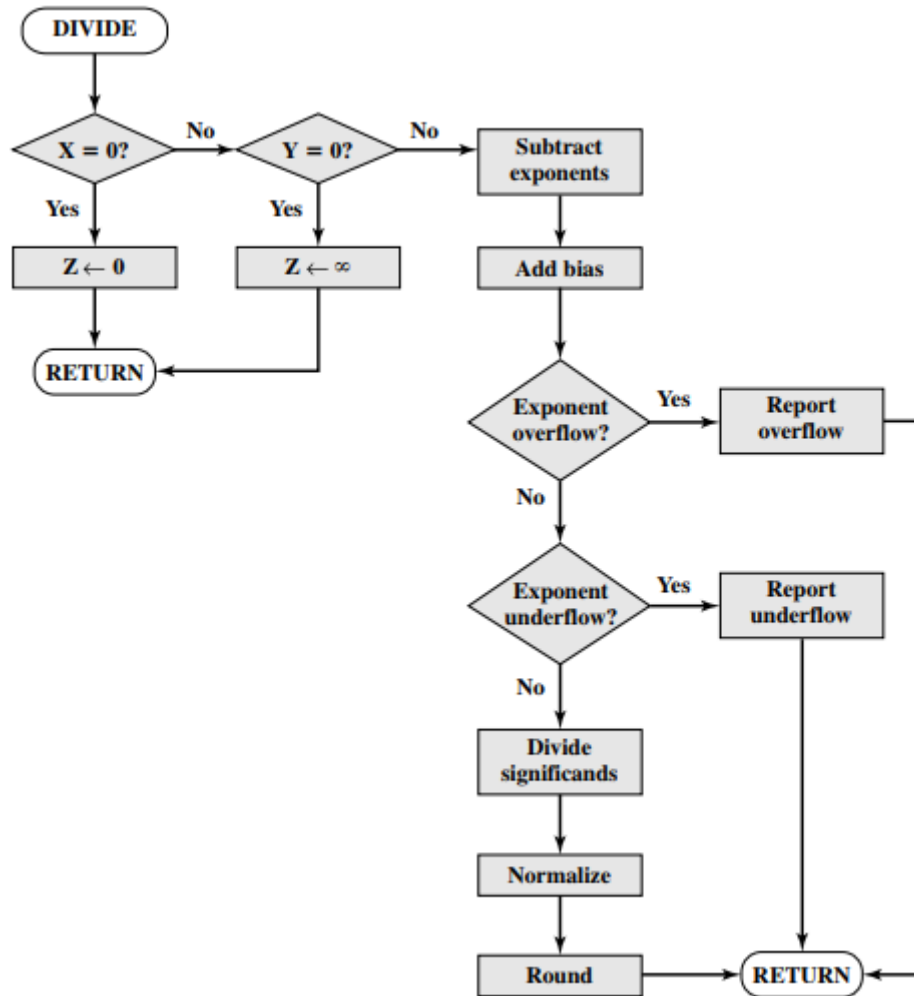
Ta thực hiện nhân theo lưu đồ thuật toán sau đây được trích từ Chương 9, sách Computer Organization and Architecture Designing for performance (8<sup>th</sup> edition) của William Stallings.



- Ta có thể tóm gọn sơ đồ trong 4 bước chính:
  - Kiểm tra phép nhân với NaN, Infinity và 0.
  - Tính toán phần mũ
  - Thực hiện nhân phần định trị
  - Chuẩn hóa kết quả (normalize result).

### 3.3.4. Toán tử chia:

Ta thực hiện chia theo lưu đồ thuật toán sau đây được trích từ Chương 9, sách Computer Organization and Architecture Designing for performance (8<sup>th</sup> edition) của William Stallings.



- Ta có thể tóm gọn sơ đồ trong 4 bước chính:
  - Kiểm tra phép chia với NaN, Infinity và 0.
  - Tính toán phần mũ
  - Thực hiện chia phần định trị
  - Chuẩn hóa kết quả (normalize result).

#### **4. PHẠM VI BIỂU DIỄN CỦA CÁC KIỂU DỮ LIỆU ĐÃ THIẾT KẾ**

##### **4.1. Kiểu số nguyên lớn QInt 128 bit:**

Do lưu trữ QInt trên 128 bit, nên phạm vi biểu diễn của QInt là từ  $-2^{127}$  đến  $2^{127} - 1$ .

##### **4.2. Kiểu số chấm động chính xác cao:**

Kiểu QFloat được lưu trữ bởi 1 bit dấu (s), 15 bit phần mũ (E), 112 bit phần định trị (F).

Ta xét:

- Số chuẩn:

- Min (+):  $1.[112 \text{ bit } 0] \times 2^{-16382} = 2^{-16382}$
- Max (+):  $1.[112 \text{ bit } 1] \times 2^{16383} = (2^0 + 2^{-1} + 2^{-2} + \dots + 2^{-112}) \times 2^{16383}$   

$$= (2 - 2^{-112}) \times 2^{16383}$$
  

$$= 2^{16384} - 2^{16271}$$
- Min (-):  $-1.[112 \text{ bit } 1] \times 2^{16383} = -(2^{16384} - 2^{16271})$
- Max (-):  $-1.[112 \text{ bit } 0] \times 2^{-16382} = -2^{-16382}$

- Số không chuẩn:

- Min (+):  $0.[111 \text{ bit } 0]1 \times 2^{-16382} = 2^{-112} \times 2^{-16382} = 2^{-16494}$
- Max (+):  $0.[112 \text{ bit } 1] \times 2^{-16382} = (2^{-1} + 2^{-2} + \dots + 2^{-112}) \times 2^{-16382}$   

$$= (1 - 2^{-112}) \times 2^{-16382}$$
  

$$= 2^{-16382} - 2^{-16494}$$
- Min (-):  $-0.[112 \text{ bit } 1] \times 2^{-16382} = -(2^{-16382} - 2^{-16494})$
- Max (-):  $-0.[111 \text{ bit } 0]1 \times 2^{-16382} = -2^{-16494}$

- Vậy phạm vi biểu diễn của QFloat là:

$$[-(2^{16384} - 2^{16271}); -2^{-16382}] \cup [-(2^{-16382} - 2^{-16494}); -2^{-16494}] \cup \{0\} \cup [2^{-16494}, 2^{-16382} - 2^{-16494}] \cup [2^{-16382}, 2^{16384} - 2^{16271}]$$

- Ngoài ra QFloat còn có dạng Infinity / -Infinity (phần mũ toàn bit 1, phần trị bằng 0)
- Nếu vượt ra ngoài các khoảng trên, thì QFloat sẽ biểu diễn dạng số báo lỗi NaN (phần mũ toàn bit 1, phần trị khác 0).

## 5. GIAO DIỆN NGƯỜI DÙNG

Giao diện được thiết kế trên C++/CLI support và .NET Framework 4.5.1. Chính vì thế, để chạy chương trình, cần cài đặt 2 components này (từ Microsoft Visual Studio 2017 về sau).

### 5.1. Chế độ QInt:

- Người dùng có thể nhập đầu vào ở hệ Bin, Dec hay Hex.
  - Ở chế độ Bin, các phím nhập số sẽ bị vô hiệu hóa ngoại trừ 0 và 1.
  - Ở chế độ Dec, các phím số sẽ được mở từ 0 đến 9.

- Ở chế độ Hex, các phím số và phím chữ (A đến F) sẽ được mở.
- Các phím thực hiện chức năng sẽ được mở dù ở hệ cơ số nào.
- Phím +/- để biểu thị số âm hoặc dương ở hệ Dec.

## **5.2. Chế độ QFloat:**

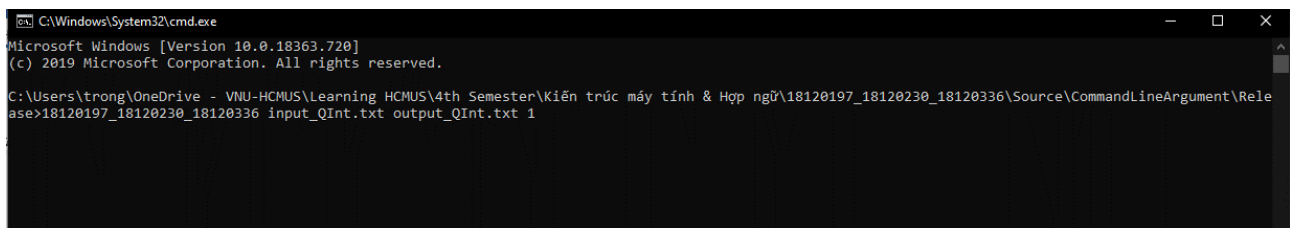
- Tương tự như QInt, người dùng có thể nhập đầu vào ở hệ Bin, Dec.
  - Ở chế độ Bin, các phím nhập số sẽ bị vô hiệu hóa ngoại trừ 0 và 1.
  - Ở chế độ Dec, các phím số sẽ được mở từ 0 đến 9.
- Các phím chức năng chỉ được mở phép cộng, trừ, nhân, chia (không có phép xoay, dịch, AND, OR, XOR, NOT,... như QInt).
- Phím +/- để biểu thị số âm hoặc dương ở hệ Dec.
- Phím . cũng được mở để biểu diễn số thực.

## **6. GIAO DIỆN CHƯƠNG TRÌNH ỨNG VỚI TEST CASE**

### **6.1. Tham số dòng lệnh:**

#### **6.1.1. Kiểu dữ liệu QInt:**

Kiểu dữ liệu QInt với 169 testcase bao gồm các yêu cầu, trường hợp như yêu cầu đề bài.



```
C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.18363.720]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\trong\OneDrive - VNU-HCMUS\Learning HCMUS\4th Semester\Kiến trúc máy tính & Hợp ngữ\18120197_18120230_18120336\Source\CommandLineArgument\Release\18120197_18120230_18120336 input_QInt.txt output_QInt.txt 1
```

Input:

Trường ĐH Khoa Học Tự Nhiên TPHCM | Khoa CNTT Trang 21 / 33

[illegible]

```

1362 2 000011010000101111100100000100001010001100001000101010001100010000011001010100110110011001010010 ^
1363 11011111101010101111101001000011100000000101000101111011110010101001001001100010101100000110001010001010001
1364 2 0111000111011000001011001010100011001100100100110001011010100100000010100100110011000010111010100001010111 ^
1365 11100011100000000100100100000100010011000010111110011001010000100010110000010100010101000001100000010101101110001110110
1366 10 ~ 8693247699586933108540
1367 16 ~ 9FE8454F5FF61EC181F6986F9B9E0B
1368 16 ~ F1D351858F91488E70DA74EE717563
1369 16 ~ C9AD1C5E6165FEE2CD
1370 16 ~ CF55F2913C15FE8B85E8AF27AD36
1371 16 ~ 1C75C2053434570DA13BAEC3C
1372 16 ~ 47B5449092CF567AAC38AE48D6F831
1373 2 ~ 1000111001110010100000111110010100000010100101001010101101110110111110010011010010100001110101100010011110
1374 2 ~ 1011010101011011100011100000010001010101110010111001001101101101100100
1375 2 ~ 00101011110000101001001000100010010010101010001100011110011001001011100001
1376 10 23264078186954652670106142 << 8
1377 -48090253898717506892192568220146 << 5
1378 10 3036 << 9378
1379 10 6172960057076055041298954 << 5200
1380 16 D49ECDFBD4CF3168F1CC739 << 9
1381 16 FC893FC29DB2B01DC5EB7929375 << 2
1382 16 46C5E959FB08A47CE8D17FD785E6 << 8
1383 2 010010001010100101101111000100101010010000101110010000101111010001100010010111110001011101011101111000100 << 9
1384 -6529624872154135214755416638567 >> 67
1385 10 42171454621913342120647207 >> 2
1386 16 045A15E6B27DB33B0AF210E210A313 >> 1
1387 2 10010000101000110111001100110101100100110101101100100110111100110 >> 1
1388 2 1100000101110110000010000 >> 11
1389 2 1111100111011110110000101011010110111010100000111000100110101001101100011 >> 8
1390 16 A943D ror 24
1391 2 11001010000001010000101111010000000011110011000 ror 3
1392 2 0101010111001000001010001011001011010100010100100010011110111111011110100101010000100001111010 ror 45
1393 2 10010101000100110000111100010011100101010100010011001000011100000001101010110100001110110 ror 32
1394 10 37053621 rol 45
1395 2 1110111110010000010100010110010110111011010010100101001001111011111101111101001010100001000011110010 rol 20
1396 2 010101011110010000010100010110010110110101000101001000100111101111110111110100101010000100001111010 rol 45
1397 -42667378749 % -743094

```

Output:

```

1 -5916556177087
2 67744706553877466985193610638133488104
3 -80814234504046499283663537294945540079
4 132816630614774637197058216469153342572
5 5685379816107727195046084
6 FC5380D4780BBFBAE920DE3900
7 73AAE6BAF70B37315
8 E198AFCB0EDEF6AB4A41A21
9 D7E3632E5FE6A2FB3CA5484501223D4E
10 A38B3EB51C53B36A4E500CE5EF70F64B
11 6B2D607DE47AC8AFBF3B91023F52FB13
12 DEDCF1C2500F54BDF269BE55167B6F9
13 369F8FCD308CF2312C4226
14 789BB822D74DF7772816165
15 77ED657828A3CFD828A6B0C53
16 FFFFEBD150FF21269C825F2AB14466A8
17 2BE85DD9C9F5A2DF7C147C5
18 EB649E8E3029287A6D1D376B2
19 FFFFFFFCBFF0080D5F005816ECE3650627
20 FFFFFFFEB32D4B8BB30F342F636CC2D
21 20F07C88644F366206
22 455EA8D5A63ABD22C17E5BD04
23 100100101110000101101010100001111100010100010110011111110
24 110110000110101010011010000001011000001111111010101
25 111111111111111111100110011011011010000011001100010111001101011011100010100101101101110111101111
26 100100000100010000001111110110000110010111000101
27 11010110000111100111010100101101101010010110000100001000101
28 111111111111111111111111100100010110011000010111111111111011110000000000011000110110100101101110000001001000011111011
29 100010111111001101000
30 1011010001011100111100100010101111110100011101000001010
31 110111100101011101001010010001111001011010001100011100110010011011011101000001000000
32 10111011010100011001100100001110011001111011111010010100010001111
33 100000111100111011010011001010101100110111101101010000100010100000110111000101101101101010111010110100110101011
34 1101111100110001000001010010101100001010110011001111011000000110111000000110000011001000110011100101101101101
35 101010000101110000001001100110101010110001100010001011011000010100001010000110011101101000001011110110100000001111
36 1101101010010011010110101101101001000110101000001100101100101111101
37 10000001100100001

```



Trường ĐH Khoa Học Tự Nhiên TP HCM | Khoa CNTT

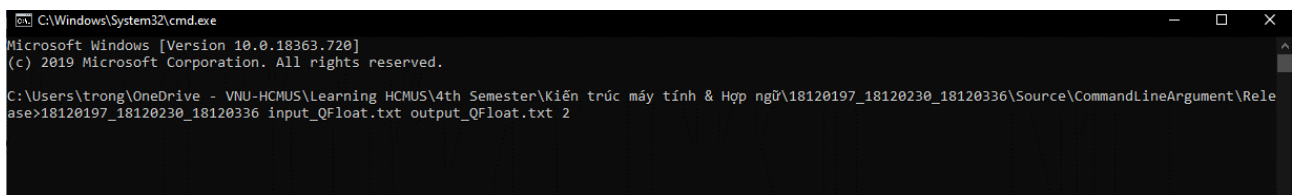
Trang 24 / 33



[illegible]

### 6.1.2. Kiểu dữ liệu QFloat:

Kiểu dữ liệu QFloat với 127 testcase bao gồm các yêu cầu, trường hợp như yêu cầu đề bài.



Input:



Trường ĐH Khoa Học Tự Nhiên TP HCM | Khoa CNTT Trang 27 / 33

Output:

[illegible]





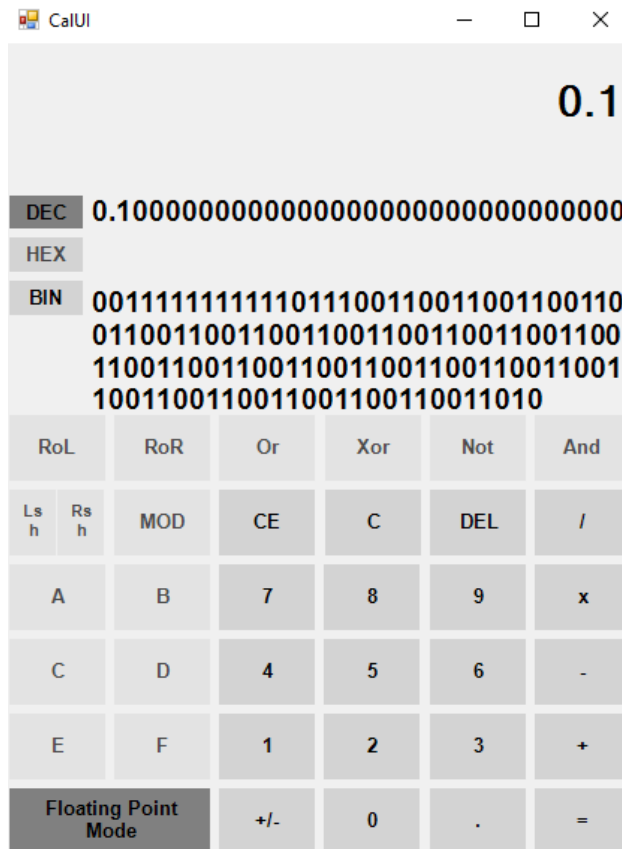
Trường ĐH Khoa Học Tự Nhiên TPHCM | Khoa CNTT

Trang 30 / 33

## 6.2. Giao diện người dùng:







## 7. ĐÁNH GIÁ MỨC ĐỘ HOÀN THÀNH

Kiểu dữ liệu / Công việc chính	Công việc	Mức độ hoàn thành
QInt	Nhập, xuất	100%
	Chuyển đổi giữa hệ cơ số a sang hệ cơ số b (với a, b thuộc {2, 10, 16})	100%
	Toán tử cộng, trừ, nhân, chia, <b>chia lấy dư</b>	100%
	Các toán tử so sánh và gán (<, >, <=, >=, ==, !=, =)	100%
	Các toán tử AND, OR, XOR, NOT	100%
	Toán tử dịch trái, dịch phải, xoay trái, xoay phải	100%
QFloat	Nhập, xuất	100%



	Chuyển đổi giữa hệ cơ số a sang hệ cơ số b (với a, b thuộc {2, 10})	100%
	Toán tử cộng, trừ, nhân, chia	100%
<b>Giao diện</b>	Thiết kế giao diện Calculator ở 2 chế độ QInt và QFloat	100%
<b>Tổng thể đồ án</b>		<b>100%</b>

## 8. TÀI LIỆU THAM KHẢO

- [1] Slides bài giảng Kiến trúc máy tính và hợp ngữ, Thầy ThS Phạm Tuấn Sơn, trường Đại học Khoa học Tự nhiên.
- [2] Quadruple-precision floating-point format: [https://en.wikipedia.org/wiki/Quadruple-precision\\_floating-point\\_format](https://en.wikipedia.org/wiki/Quadruple-precision_floating-point_format)
- [3] Floating-Point Arithmetic, A. Yavuz Oruç Professor, UMD, College Park: <https://user.eng.umd.edu/~yavuz/teaching/courses/enee446/lecture%20notes/lecture4467N8N9.ppt.pdf>
- [4] Computer Organization and Architecture Designing for performance (8<sup>th</sup> edition), William Stallings.
- [5] Visual C++ Calculator Tutorial, DJ Oamen: <https://youtu.be/zBBe9oKU7jk>